AMENDMENTS TO THE SPECIFICATION

In the Specification:

Please replace the title on page 1 of the specification with the following amended title:

Title: SYSTEM AND METHOD FOR MODELING BUSINESS WORKFLOW PROCESSES EMPLOYING A SCHEDULER USER INTERFACE [[UI]]

Please replace the paragraph beginning on page 1, line 31 with the following amended paragraph:

A fundamental concept of workflow analysis is that many business processes can be interpreted as a sequence of basic transactions called workflows. Workflows have a customer, a performer, and conditions of satisfaction. The customer and performer are roles that participants can take in workflows. In addition, workflows can have observers. In conventional business workflow systems, a transaction comprises a sequence of operations that change recoverable resources and data from one consistent state into another, and if a failure occurs before the transaction reaches normal termination those updates will be rolled back or undone. ACID transactions control concurrency by isolating atomic transitions against each other to create a serializable schedule by delaying updates until committing of transactions. This isolation limits granularity viewed by an observer to the size of the parent ~~transactions,~~ transactions until all child transactions commit within a parent transaction. Therefore, application specific programs cannot be invoked and monitoring of transactions by [[user's]] users cannot be performed based on any actions occurring within a transaction, until the transaction fails or commits.

Please replace the paragraph beginning on page 2, line 14 with the following amended paragraph:

The Unified Modeling Language (UML) defines a standard notation for representing flow charts for business workflow processes. However, the UML notation cannot be reduced to a programming language for employing complicated workflow processes. Current business workflow software systems provide scheduling software that requires binding within the scheduling to couple the schedule to real world applications and technologies. Conventional schedules ~~requires~~ require code to couple the components of the schedule to application program interface (API) objects and/or server objects for interfacing the schedule with systems within each business or department involved in the business process. These types of schedule software require sophisticated programmers ~~in implementing~~ to implement the software for a given business workflow model. Furthermore, these types of schedule software require modification of each schedule for different technologies.

Please replace the paragraph beginning on page 3, line 2 with the following amended paragraph:

A user is provided with a GUI schedule interface, which allows the user to create a schedule on a first side of the GUI and to define bindings on the other side of the GUI. During creation of the schedule, the scheduler program prohibits the user from creating a schedule that will deadlock the schedule by checking correctness of the schedule flow. In creating the binding, the GUI schedule program provides prompts for specifying interfaces and methods of the components being bound. A data flow connection sheet is provided based on the schedule messages and the binding component interfaces and methods. The data flow of messages is then defined by simply connecting the message ports to binding component interfaces to facilitate proper data flow between entities. It is to be appreciated that the separate data flow sheet could be combined into the GUI scheduler interface or provided via a pop up screen after defining binding component interfaces and methods.

Please replace the paragraph beginning on page 13, line 19 with the following amended paragraph:

The present invention will now be described with respect to a rule set associated with the GUI scheduling software 10. A dynamic connector tool ~~connector~~ can be enabled and resides on a toolbar menu and a flow chart stencil. The present invention supports one ~~construct, context,~~ construction context with two applications: roles and transactions. The UI includes one shape for roles and one similar shape for transactions, that each translate to contexts in XML. Figure 7a illustrates the nesting of roles and transactions. The role and transaction shapes share most features, such as Cut/Copy/Paste behavior, wiring control flow into and out-of components, RMA's and association. Role shapes have communicative connections and role ports. Transaction shapes can have compensation and catch pages associated with the transaction. Compensation refers to routines invoked on other transactions upon a failure of ~~a~~ given transaction, while catch refers to routines invoked on a failed transaction. Transactions can only be nested two levels. Figs. 7b and 7c illustrate valid transaction nesting, while Fig. 7d illustrates invalid transaction nesting. There is no limitation on the nesting of roles. Fig. 7e illustrates valid role nesting. Roles and transactions can be nested within each other as illustrated in Figs. 7f and 7g. The only limitation being that transactions can only be nested two levels.

Please replace the paragraph beginning on page 14, line 4 with the following amended paragraph:

Fig. 8a illustrates the role shape highlighted, while Fig. 8b illustrates the role shape not highlighted. The role shape has sharp corners, while the transaction shape has rounded corners to differentiate the shape of a role from the shape of a transaction. Once a role is annotated with role ports, it easily differentiated from a transaction. Like the transaction shape, the role shape has only one connection point located at the top center of the shape. The system allows only one incoming control flow to be attached to the role's connection point. This forces the semantics of having only one point [[on]] of entry into the context of the role. There can be any number of ~~controls~~ control flows leaving the role. A control flow leaving a role can be attached to any shape [[that]] to which an action shape can be attached. Control flows that exit a role do so

wherever the user desires. The incoming control flow rule is that only one control flow may enter the role. The control flow enters the role at the top center of the role. Maintenance of this rule requires some legal applications of roles and others that will be rejected by the system. Fig. 8c illustrates some examples of "legal" role groupings. Fig. 8d illustrates a role before association and Fig. 8e illustrates a role after association. The role shape can have different changes upon association depending on whether or not there is an incoming control flow. If there is an incoming control flow, then that control flow will have to be re-connected to the role's connection point, and another connector shall be added, attaching the role's connection point to the action state that was originally attached to the incoming control flow. There is no special processing for the control flows that exit a role. Not all role enclosures are legal. Fig. 8f illustrates an illegal role association. There are two control flow lines entering the role. If the user attempts to associate the role with the enclosed shapes, the system will present an error message. This role can't be associated because there is more than one control flow path into the role.

Please replace the paragraph beginning on page 18, line 3 with the following amended paragraph:

Transactions have two additional properties that roles lack, that is enable compensation and enable catch. These properties can be enabled independently. Fig. 15 illustrates a pop-up window (GUI) showing transaction properties. If either catch or compensation is enabled, then a new view is created, either "compensation for [transaction-name]" or "catch for [transaction-name]." These views are represented by tabs at the bottom of the page, similar to Business Logic and Data tabs (See Fig. 2). Since transactions can be nested, an interior transaction may have both a catch and a compensation transaction. Those transactions themselves may have both a catch and a compensation transaction. The Business Logic page, which contains root transactions, catch, and compensation pages all look and feel the same, but have completely independent control flows. However, they all share the same separator bar, ports, message, and bindings. Like the main Business Logic page, each catch or compensation page is created only with a begin shape on the flow chart side. Each page can be viewed simultaneously using windowing features. Cut / Copy / Paste [[is]] <u>are</u> possible between all views (business logic and

each compensation or catch page). All workflows will start out with an undeletable "Workflow Catch Page" which is the default error handler for the entire workflow. In this case, there will be three initial pages, Business Logic, Data flow and Workflow Catch. Table 2 illustrates the properties associated with transactions in various states.

Please replace the paragraph beginning on page 20, line 13 with the following amended paragraph:

Fig. 16a illustrates a switch shape that is provided for providing a decision component for the GUI scheduling program. The switch shape can map directly to an XML programming language component. The switch shape contains rule shapes. Rule shapes are uniquely named. Default rule names will be "Rule x" where x is an integer ~~1,2, etc~~ 1, 2, *etc.*, similar to the auto naming of ports, actions, transactions, etc. Multiple switch shapes can contain the same rule. A specific rule will only be used once within a specific switch. Cleanup will delete rules that are no longer used in any switch shapes. Cleanup is a term used to refer to two menu options under the Tools menu. Each option deletes shapes. Each Switch shape always has one, un-editable, "Else" rule shape. When a switch shape is originally dragged onto the page, the only rule will be Default. A user will need to edit properties on the shape to add additional rules, such as the three new rules added to the switch shape illustrated in Fig. 16b. Each switch shape will size itself according to the longest rule that it contains. Each rule will be the same size. The shape has one connection at the top of the diamond where the flow enters the shape. Each rule can have one connection coming out of either the left or right side. A user can wire the rule by dragging the control handle located at the left or right edge of the rule. Redrawing a rule outbound connection, from either side, will not require deletion of the previous connection. The newly drawn control flow connection will replace the old one.

Please replace the paragraph beginning on page 22, line 27 with the following amended paragraph:

Every message has two system fields ("Result Status", int) and ("Sender", string). The user shall not be able to enter a default value for the "Result Status" or Sender field. In the Field display section of each message editor, Name and Data Type are not editable. Data Types are typically displayed as XML types, regardless of message editor. Field names are unique per message. For the name, each message editor has a combobox containing all existing messages to enable the user to say that this message is exactly another message. Each message editor will have a rename button to the right of the combobox to allow the user to rename the currently selected message. This rename/combox will be the same whether selected from the data or business logic pages. Message names are displayed as a textbox, on both the business logic and data pages. Unique name validation will be done when the user presses the "OK" button. This editor will allow the user to specify two schemas (.XML files). The Message fields frame appears identical to the Method message editor. The names and data types are populated from the top-level XML Elements in the Internal Schema specified in the Message Schema frame. The user will be able to select from all transactions on the page. With COM and Method bindings, the fields of the messages are automatically set to either the [in] or the [out] arguments of method they are pointed to in the port. With MSMQ bindings, the user will have to go into the message ~~editor~~ editor, specify an XML schema, and select nodes in the tree to populate the fields. With CALL bindings, the user will have to go and specify the ports, messages, and transactions being passed into the called schedule.